EXHIBIT U

Reliabot\eVF Comparison and Evaluation

COPY

Summary

This document is intended to provide an objective examination of Reliabot and a comparison to eVF. The examination is broken down into a number of categories which cover everything from Vision Science on the low level end, to Maintenance and Support on the high level end.

1. System Price

- See Price comparison excel sheet

2. Vision Guidance Science

See Appendix A for comprehensive evaluation

Reliabot	eVF
• Simple vision guidance algorithms that use basic linear algebra	• Advanced algorithms based on minimization, cost-funtions, least squares, etc.
• Some of the algorithms are hardcoded to match specific geometric shapes	• Generic registration based algorithms, allows for models of any 2D or 3D shape or surface.
• The stereo algorithms have limitations in the numbers of geometric entities that can be matched	• Advanced stereo matching techniques (dense stereo, feature based)
• 2.5D uses two cameras and a minimum of one feature is required. More expensive approach and longer setup time.	• 2.5D uses one camera and a minimum of two features is required
• 3D uses two cameras. More expensive approach and longer setup time.	• 3D uses one camera. Cheaper, faster, less setup time, smaller tool footprint.
• Bin Picking has never been done randomly per John Neilson. The piston application was never sold.	• Supports Bin Picking for organized parts in a bin • Supports Random Bin Picking for parts in a bin.
• Camera calibration does not calculate lens distortion. This results in losing system accuracy and reduces feature finding robustness since the image is processed as a distorted image.	• eVF calculates lens distortion. More accurate.
• There is no automatic calculation of a camera to robot transformation – the user has to touch points in order to relate the part / camera with a robot coordinate system • The calibration 3D algorithm uses permutation to find the 'best' calibration (no use of cost functions minimizations)	• Removal of human subjectivity from calibration process results in higher accuracy.
• There are no quantitative algorithms to determine the accuracy, tolerances, robustness of any vision algorithms	• Quantitative algorithms to determine the accuracy, tolerances, robustness of any vision algorithms

Conclusions

Reliabot employs undergraduate level linear algebra to solve most pose estimation problems. Where some problems could be solved using more sophisticated mathematical techniques, the Reliabot approach is usually to add more cameras, adding cost, set-up time and additional points of failure to the solution. The lack of rigorous approaches to camera calibration (i.e. accounting for lens distortion), or quantitative methods to evaluate accuracy will make it very difficult to demonstrate that an installation has met, or is even capable of meeting a solution's required specs. Furthermore, the emphasis on simplicity limits the domain of solvable problems open to Reliabot and reduces the barrier to entry for competing solutions within that domain. By contrast, eVF uses the best mathematical tools available to compute the most accurate pose possible with the fewest cameras necessary, and can **quantify** its confidence, taking into account all sources of error. It can also be used to diagnose the most significant source of error so that it can be mitigated or corrected. The automotive comparison is apt: if Reliabot's science is a 4 cylinder Civic, eVF is a F1 racer, and no amount of flashy add-ons will put the two in the same class.

3. Software Architecture and Engineering

See Appendix B for comprehensive evaluation

Reliabot	eVF
• Basic architecture is based on the concept of the CF file. CF files define the run-time configuration of the Reliabot UI, vision guidance and device interaction (robots, cameras, etc). CF file syntax consists of approx. 160 distinct commands. Each command is indicated by a three letter acronym (e.g. DAT or CA3) and will have one or more numerical or text arguments, subcommands, etc. Documentation is available for a handful of commands.	• The eVF analogue of the CF file is the eVF Workspace. The Workspace is configured using the eVF user interface and, with minor exceptions, is encoded such that reverse engineering is very difficult. More significantly, the eVF Workspace stores configuration parameters but no significant run-time logic related to vision science.
• Solution creation is done using multiple different programs. Cognex VisionPro, Reliabot, Text Editor for CF File. Note: 60% of the vision solution must be done in the CF file. 20% in the Cognex VisionPro software and the remaining 20% of the vision solution can be edited in the Reliabot GUI. • Switching back and forth is not seamless, causes confusion	• 100% configured through one graphical interface. (Point and click solution development). Third party tools are integrated into eVF.
• New versions of Reliabot were produced for each solution that was developed. Each version was heavily customized for the solution in question. Customizations may have involved adding or altering software code, or hand-tweaking the contents of CF files. Most importantly, new versions were often created by copying the entire directory of source files and/or an existing CF file. **This is a major obstacle to scalability of the product.**	• The eVF development model involves strict version control, a regular develop-test-release cycle and ongoing vigilance as to the quality of the finished product. eVF has benefitted from a rigorous testing regimen that focuses on both the quality of the vision science and end user usability. As a result, while the user is prevented from examining the internals of a workspace, the user-friendliness of the interface, coupled with extensive end-user documentation, enables them to solve minor configuration issues without relying heavily on Braintech support.
• Where CF files represent the configuration state of a particular solution, the software code represents the Reliabot program itself. Like CF commands, most of the Reliabot modules are represented by three-letter acronyms (e.g. ado, ape, se2). There does not appear to be a 1-1 correspondence between modules and CF commands. Most of the source files contain some short documentation describing the module and revision history. For a complex, large-scale software system, Reliabot is relatively small. A great deal of the vision science is out-sourced to third party libraries such as Cognex VisionPro, and in addition, considerable solution logic is contained in the CF files.	• eVF development based on the concepts of object-oriented programming and modularity. This makes it possible to expand the capabilities of the system to support new hardware devices and new vision guidance methodologies. However, the eVF code base is very large, consisting of approximately 2500 source files containing 580,000 lines of code. The size and complexity of the code base makes it impossible for any one developer or scientist to be familiar with everything. This can present difficulties when new features are added, and as a result new feature development is typically a team effort

Conclusions

The outstanding differences between the eVF and Reliabot architectures are that eVF is well-documented with a strong object-oriented methodology, and has benefitted from a focus on developing, testing, and maintaining a single software product on which all end-user solutions are based. There is one and only one eVF code base and new versions of eVF aim for backwards compatibility with previous commercial releases. By contrast, there will be considerable obstacles in

identifying a stable code base, and building, testing and extending a core software product from the current Reliabot code. We believe that these are critical capabilities if Reliabot is to thrive as a provider of vision guidance solutions.

4. Maintenance and Support

Reliabot	eVF
• Could not find step by step documents to create a solution. This is required by current customers such as Gudel, PSDI and ICI.	• Fully documented system. Provides step by step instructions.
• Training new users with a lack of documentation is going to be difficult. I need at least another 8 weeks to complete Reliabot's documentation to support customers.	• Fully documented system. Provides step by step instructions.
• The CF creates problems with users being able to scale vision solutions because it is difficult to learn, add new models and difficult to remember a cryptic CF file	
• Getting a solution from Gudel on home PC without a frame grabber required John to drastically modify the customer's original solution. (I.e. Cognex VPP files -disable the frame grabber, add offline file components). It is very complex to troubleshoot a customer's vision solution. You also have to change everything back before sending the new vision solution to the customer.	• Perform a restore in seconds without having to modify the original solution
• According to John Neilson only two companies can integrate Reliabot. These two companies are Utica Enterprises (17 systems) and CEC Controls. They had gained experience from working on several Reliabot projects. The vision manager of Utica does not want to work with Adil anymore.	
• Shafi CD Software Installation is very complex and difficult to document. John Neilson said after you install the Shafi software it only installs the basic version. You then have to download WinZip and install it, then navigate to a special folder which contains 74 different current Shafi program versions based on communication method and robot type. The file has to be manually extracted to overwrite the installed program. He mentioned that every new customer wanted something different, so a different flavor of the software was created for that particular customer rather than add the new features to the main software branch and maintain one main version. The maintenance of the software got out of control. If the customer asked for a replacement CD, how do you support it? See screenshot below showing 74 different program files found.	• Run the setup once and a wizard guides the user to install all the software. Uses the same methods as installing any Microsoft project.

5. User Interface and Learning Curve

Reliabot	eVF
Solution Development • No GUI for creating entire vision solution. • Created using multiple different programs. Cognex VisionPro, Reliabot, Text Editor for CF File. Note: 60% of the vision solution must be done in the CF file. 20% in the Cognex VisionPro software and the remaining 20% of the vision solution can be edited in the Reliabot GUI. • Switching back and forth is not seamless, causes confusion	**Solution Development** • 100% configured through one graphical interface. (Point and click solution development)
• Simplicity of user interface can reduce confusion for operators. Customization is possible to create messages specific to a particular solution.	• User interface is more complex, so there are more controls visible by the operator. It is still possible to disable these controls selectively to reduce possibility of inadvertent changes.
• CF file syntax consists of approx. 161 distinct commands. Each command is indicated by a three letter acronym (e.g. DAT or CA3) and will have one or more numerical or text arguments, subcommands, etc. Documentation is available for a handful of commands.	• 100% configured through one graphical interface. (Point and click solution development)

6. Hardware Support

Reliabot	eVF
• Congex frame grabber is $2300 plus $400 frame grabber cable	• Matrox Frame grabber is $550
• Cognex Frame Grabber supports a maximum of 4 cameras.	• Matrox Frame Grabber supports a maximum of 6 cameras.
• Cognex Vision License – $3000	• Matrox Vision License $1250
• Supports any camera which can connect to a Cognex framegrabber	• Supports any camera which can connect to a Matrox framegrabber.
•	• Supports GigE cameras which do not require a framegrabber.

7. Robot Interfaces and Robot Programs

See Appendix C for comprehensive evaluation

Reliabot	eVF
• The majority of communication in Reliabot is accomplished by sending raw data over a serial connection, and having the robot programs interpret this raw data. This means that individual robots do not have specific communication interfaces. It is the robot programs that need to be created for each robot controller. The exceptions are Fanuc, which has a specific Ethernet interface, and Motoman, which uses the Motoman communication serial protocol.	• Communication in eVF has been accomplished by individual communication components for each robot controller brand. This makes it possible to read and write directly to named variables on the robot side. Each communication interface makes use of the latest standard interface available for that robot (Ethernet if possible).
• Reliabot robot program examples are available for seven robot types; ABB, Adept, Fanuc, Kawasaki, Motoman, Nachi, Staubli.	• There are four robots supported by eVF; ABB, Motoman, Kawasaki, Denso. Kuka is nearing completion.
• Could not find standard robot programs (API) for each brand of robot.	• Robot programs are created as Application Programming Interfaces that

User must create robot code to use the Reliabot data from scratch based on examples from previous solutions. Vision related code is in the form of routines that are added to perform three basic operations; Sending trigger from robot to Reliabot, receiving data from Reliabot, and offsetting points to be used for motion.	simplify the use of vision by robot programmers, and make the vision related code portable to any solution. Communication in the eVF case is not visible to the robot program and does not need to be run explicitly.
• The adjustments to robot motion based on vision are done on a point by point basis. For instance, let's say there is a point, P1 that needs to be moved to for picking a part. After vision is run, the new coordinates of P1 are sent from Reliabot. No complex operations are possible.	• There are a number of advanced functionalities that are supported for ABB(and to a lesser extent, Motoman). These include AutoCal, AutoTrain, Accutest, ReOrientation for increased accuracy, automated movement of robot to training position.

Conclusions:
The Reliabot communication method has the benefit of being able to support multiple robots from the same interface, and more robot brands have been integrated. Adding robots still requires developing robot programs to provide the other side of the communication link. The Reliabot communication interface is antiquated, slow, and very simplistic. The eVF communication interfaces are designed to provide advanced functionality and automated procedures that are not possible with Reliabot. While eVF does not currently offer as many robot interfaces as Reliabot, the path to arriving at a set of full featured and flexible robot communication interfaces will be much shorter by adding more robots to eVF, than by extending or recreating every interface in Reliabot.

8. Software Development Environment and Tools

See Appendix D for comprehensive evaluation

Reliabot	EVF
• Reliabot is coded in Visual Basic 6 (VB6, aka Classic Visual Basic). VB6, the final version of Visual Basic was released in 1998. • **Mainstream support for VB6 ended on March 31, 2005, and Extended support ended on April 8, 2008.**	• eVF is developed using a combination of C/C++ and C# (mostly C++). Newer projects are increasingly written in C#.
• Visual Basic has inadequate support of multiple threads required for advanced applications like bin picking • Performance lags that of C/C++. Probably also lags .NET. • Lack of language support implies difficulties integrating new hardware (cameras, robots, etc). • GUI development is fast and easy.	• Code is written in VC++ which supports multiple threads. I.e. takes advantages of the new multiple core processors automatically. • Fully object oriented (ease of extensibility, modern development concepts, etc). • Access to WIN32 API and C programming = major performance advantage. • Ease of integration with new hardware
• Six month learning curve for Braintech employees to learn Visual Basic	• Programmers already know VC++
• Reliabot is dependant on Cognex Vision Pro version 3.4 which is two generations old. The latest Cognex version is 5.0 with the software application to be written in .NET. Reliabot would need to be re-written in another language to take advantage of Cognex's latest software.	•

Conclusions:

Continuing Reliabot support using VB6 presents a serious challenge- lack of vendor support, lack of available development licenses, and a lack of modern programming language advances all represent red flags against developing and maintaining a large-scale software. There is no easy out in this respect, since re-writing Reliabot in a modern language would represent thousands of developer-hours. While eVF is also built on aging technology, the core C/C++ languages will be supported for decades to come, and our ability to inter-operate with C# can smooth the path to modernization. The main concern with eVF is its dependency on COM and marriage to the Windows platform- a concern that should be resolved with the roll-out of eVF 7.0 (BVGL/BVSL).

9. User Scenarios Automation

Reliabot	eVF
Camera Calibration • Not automated. (Bolt pointer to the robot, carefully train the pointer, print bulls eye target, add bulls eye to part in three locations, user jogs pointer to each bulls eye location and records x, y, z pointer coordinate, carefully enter each x, y, z value into the Reliabot software. Repeat step three times) • Subjective to operator who is performing the calibration. • It is also more time consuming which costs the plant money. • 1 hour for experienced user or several hours for non-experienced • Camera Calibration integration time takes longer as the user needs to enter the calibration offsets into the CF file. • Two camera calibrations required for 2D and 3D for a single camera • Dave Coker who works for DC Automation has experience in Reliabot and eVF. His comments on comparing both products were that Reliabot takes twice as long to perform all the integration steps.	**Camera Calibration** • Fully automatic calibration. User clicks one button. Let's show a demonstration to Pete and Rick. We can simulate it for Reliabot. • Takes only two minutes • Camera Calibration integration takes a few minutes. Just load in a standard API into each robot. • One camera calibration supports 2D and 3D for a single camera.
• Solution validation is based on moving the part in space and testing it with a pointer.	• Accutest: Vision solution validation is done automatically by having eVF move the robot, collect images, and process results. Accuracy and repeatability data is provided in standard format results files.
• Each feature position must be manually entered into the system.	• AutoTrain: Feature training is done automatically by having eVF move the robot, collect images, and process results.

Conclusions:
Some attempts have been made to simplify and automate the calibration process in Reliabot, but there is still a large amount of manual steps and subjective touching of targets by jogging the robot. eVF has been designed to almost completely remove the burden of three major configuration procedures; calibration, model training, and solution validation(accuracy testing). All the integrator needs to do is enter the relevant parameters in eVF, then press a button. All other action is performed automatically and comprehensive results are created when the operation is complete. This saves countless setup hours, and ensures that the solution is configured optimally.

10. Typical Solution Configuration Steps and Required Time

Reliabot	eVF
• 3D camera calibration – 1 hour for experienced user	• 3D camera calibration – 2 minutes for non-experienced user
• 2D camera calibration – 1.5 hours for experienced user	• 3D camera calibration – 2 minutes for non-experienced user
• Vision Solution Develop – 2 days for robot tech (from field tech Dave Coker)	• Vision Solution creation – 1 day for robot tech. (from field tech Dave Coker)
• Vision Solution maintenance i.e. Add additional models at a later time– Very difficult –Requires in-depth knowledge.	• Graphical based – should only take minutes

Appendix A – Vision Guidance Science

A try to evaluate the vision algorithms of the current Reliabot software

A list of identified source code files that contains vision and vision guidance related algorithms available in Reliabot source code:

\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\Ca1.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\Ca6.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\MO3.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\MO2.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\MSU.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\SLI.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\VB2.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\VB3.bas
\\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2\VPS.bas

The algorithms are part of "PSEUDOLIBRARY" a custom name assigned by Shafi to group and handle libraries in Visual Basic. This is a software technique available in Visual Basic, created by Shafi in 1997 and it looks it was not updated until today.
For each file the existing description (in *italics*) is copied followed by my notes in **bold**.
Based on my understanding of Reliabot structure, only the files from one solution folder (the root of \\jupiter\Software\SHAFI\SWP\RELIABOT\cognex\M2) were analyzed.

Conclusions:

It looks like the sophistication of the vision problems can be solved easily by anyone with basic engineering degree.

1. These are simple vision guidance algorithms that use basic linear algebra
2. Some of the algorithms are hardcoded to match specific geometric shapes
3. Simple linear interpolation functions
4. Simple distance functions that sort / rearrangement list of points
5. There is no automatic calculation of a camera to robot transformation – the user has to touch points in order to relate the part / camera with a robot coordinate system
6. There are 2D and 3D utility functions for operation with coordinate systems
7. Bin picking heuristics uses simple constraints and approximations to match a gpm in two or more images / cameras – accuracy could be an issue.
8. The calibration algorithms do not use advanced camera models to calculate distortions – accuracy could be an issue and maybe lens with large FOV can't be use because of distortion.
9. The stereo algorithms have limitations in the numbers of geometric entities that can be matched.
10. The calibration 3D algorithm uses permutation to find the 'best' calibration (no use of cost functions minimizations)
11. There are no advanced algorithms for SR3D, SL3D, RBP type applications and vision solutions.
12. There are no quantitative algorithms to determine the accuracy, tolerances, robustness of any vision algorithms
13. There are no automatic procedures for autocal, autotrain and accutest

Ca1.bas

This library is the new, formalized RELIABOT PC prorating engine, or the new 1D simplifed calibration engine. We start with two values A and B and have some "offsets"
*for each. We then take a new value C and calculate what the offsets *would* be for C based on if C is closer to A or closer to B. It is prorated evenly. For example, if A is 0 and B is 10 and C is 3, then C's offsets will have 70% of A's and 30% of C's.*
There can be up to 6 values in the "offset". It all depends on the CF file. The values A, B, and C, though, are single values (hence 1D calibration).
' CF file format:
' OBJ
' QUA(1) [A]
' FPA [offset1 offset2 offset3 offset4 offset5 offset6]
' QUA(2) [B]
' FPA [offset1 offset2 offset3 offset4 offset5 offset6]
'

' For more details, please refer to the "Calibration Systems" ' document (C=KJN 13 Apr 04).

This is a linear interpolation algorithm that applies to each component of a transformation c = w*c1 + (1-w)*c2

Cd6.bas

This library is the new RELIABOT PC discrete calibration
' engine (CDx). It can handle up to 6 dimensions (CD1 to CD6).
' It is assumed that the first three are linear (X, Y, Z)
' dimensions and the latter 3 are auxillary dimensions (such
' as yaw, pitch, and roll angles).
'
' The goal of discrete calibration is to determine which of
' a finite number of discrete "zones" a point falls into.
' Examples of discrete calibration:
' 1. RS Tubs geometric zone check (GZC).
' 2. Discrete fine pick adjustment (FPA discrete).
' 3. CCVRID-EV layering algorithm.
'
' CF file format:
' OBJ
' ZON(1) [x y z y p r]
' TOL [x y z y p r]
' DAT [x y z y p r]
' ZON(2) [x y z y p r]
' (etc.)
'
' CF file format Variants allowed:
' TOL can be at ZON's indentation level with an index
' specified (for that zone only) or with no index for
' a global tolerance.
' ZON can also be QUA.
' DAT can also be FPA. Or it can be omitted.
'
' CF file object description:

' ZON defines the zone point. We take the input point
' and find the nearest zone to that point.
' If TOL is specified, then, once we do find the
' nearest point, we make sure it is within this
' tolerance.
' If DAT is specified, then we return the AUDO ID
' of that object. Otherwise we return the AUDO ID
' of the ZON object.
'
' For more details, please refer to the "Calibration Systems"
' document (C=KJN 13 Apr 04).

elem_tol_check() checks if a transformation is with a tolerance of another transformation.
abs(c1-c2) > tol then transformation is outside of tolerance interval. In case of 3D (as this is the case) the above formula is not valid. The proper checking is:
(A*B-1) be checked and not simple subtractions of each component.
Zone_find() find the correct zone for a given input point.
This is a 2D algorithm that finds in which area the given point is located by calculating the minimum distance to a list of points.

Lig.bas

' This pseudolibrary implements linear regression to create
' a best-fit line through a set of 2D points. The equation
' used is:
'
' y' = a + bx
'
' Given a set of (x,y) points, an equation is created to
' predict y based on x.
'
' This pseudolibrary can also be used to compute other
' statistics, such as mean or standard deviation.
'
' Note: For ease of user interface, you give one (x,y) to the

' computation engine at a time. For this reason, you can't
' compute two equations simultaneously (i.e., you can't have
' two threads try to use this module at the same time).
'
' This module created 8:30pm on April 29, 2002.
' LIG = Linear 'Gression

It is not used anywhere else in the code.

MO3.bas

' This module is a "next generation" geometric modeling (MO3)
' library, which includes algorithms to pick out a valid model
' from an assortment of 3D points. It was created on May 21,
' 2003. Currently it uses all new algorithms instead of the
' 1996 wireframe modeling.
'
' Note: The utilities utilize the following L3D "3D point list"
' object structure:
'
' L3D
' XYZ type [x y z]
' XYZ type [x y z]
' XYZ type [x y z]
' (etc.)
'
' IMPORTANT: The list above is an ordered list.
'
' Modeling currently involves grouping candidate points
' together, then rebuilding the L3D list when done. In the
' process, a 3D point may then appear more than once in the
' list -OR- it may be excluded as spurious.

3D points management and utility functions based on labeling and distance and other criteria (lowest x first, highest x first ....)
dist_match() matching function for group candidates based on distance functions (dist < tol).

Make_frame() build a frame using different input parameters (from one, two or three points, 4 and 8 points) In case of one point the frame as the same orientation as the coordinate frame where the point is defined in.
Frame utility functions – frame-standardize – add two 3D transformations.
Frameize() – from a list of points calculate / collapse to a frame – this is not a PCA type implementation (center of mass, inertia axes...) it is a simple 1, 2, 3, 4, 8 frame calculation.

MSU.bas (bin-picking)

```
' [* = needed for multi-camera matching]
'
' This module is a Matching/Sorting (Screening) Utilities (MSU)
' library, created March 19, 2003, in preparation for Nachi
' Bin Picking (our first 3D demo involving matching similar
' fiducials among multiple cameras). It includes some
' more heuristic, simpler, single-camera matching techniques
' which can be used in simpler applications. Multi-camera
' matching algorithms added May 20, 2003.
'
' Note: All matching/sorting utilizes the following L2D
' "2D point list" object structure:
'
'  L2D
'    OBJ(1) "Object #1" [submodel]
'      DAT(1) "Cam 1" [u v ang score id zfix scalex scaley]
'      DAT(2) "Cam 2" [u v ang score id zfix scalex scaley]
'      DAT(3) "Cam 3" [u v ang score id zfix scalex scaley]
'    OBJ(2) "Object #2" [submodel]
'      DAT(1) "Cam 1" [u v ang score id zfix scalex scaley]
'      DAT(2) "Cam 2" [u v ang score id zfix scalex scaley]
'      DAT(3) "Cam 3" [u v ang score id zfix scalex scaley]
'    (etc.)
'
' IMPORTANT: If there are n OBJ objects, then all n must exist.
'
' The items in the list are sorted by algorithms in this module.
```

' This involves swapping DAT lines among objects. Submodels
' may or may not be taken into account depending on the
' algorithm. Great care is taken to account for degenerate
' cases (either with -99999's for points or missing points).
'
' With multi-camera algorithms, the idea is that the U-V
' coordinates for an object may be mixed up; the algorithms
' are responsible for putting the right U-V coordinates with
' the right vision objects.
'
' New, 17 November 2005: The following syntax is allowed:
'
' L2D
' OBJ(1) "Object #1" [submodel]
' DAT(1) [u v ang]
' XAT(1) "Cam 1" [u v ang score id zfix scalex scaley]
' XAT(1) "Cam 1" [u v ang score id zfix scalex scaley]
' XAT(1) "Cam 1" [u v ang score id zfix scalex scaley]
' DAT(2) "Cam 2" [u v ang score id zfix scalex scaley]
' DAT(3) "Cam 3" [u v ang score id zfix scalex scaley]
'
' This structure allows DAT(1) to be a representative for a group
' of locations having average coordinates. This allows, for example,
' the hmc_bin() algorithm to work with one point per group for
' easier matching rather than match (and mix up) individual points.
' Once this operation is complete, the XAT's should be returned to
' normal DAT format immediately. The conversion to and from both
' formats is handled via sc_rebuild() (and by sc_ungroup()).

Typecomp() – the objects in Shafi Reliabot bin-picking have a type attached to them as model identifiers. Type comparison means:
PMAlign1 PMAlign2 IF PMAlign1 = PMAlign2
PMAlign1 -PMAlign2 IF PMAlign1 <> PMAlign2
Psort() – sort clockwise an array of 2 points; 3D points are projected on X-Y plane for sorting.
Sc_abab() – single-camera sorting technique – sort 2d points to get grouped in a diamond shape.

Sc_garden() – hardcoded arrangement of points from 2 cameras.
Camera sorting techniques – all hardcoded – *if a registration type algorithm would be available, there would not be a need for the hardcoded type matching.*
The sorting algortithm solve specific point configurations and for this reason the usage is limited.
Hmc_bin() – using camera physical arrangements to reduce / transfer the search area from one camera to another. It looks like the algorithm pairs matches from multiple cameras based 2D transformation from the center of bin. A better approach would be to use stereo or multiple camera constraint searches.
Simple stereo matching for specific image features are provided:
a. one straight line per camera (mc_cylinder) – the algorithm is limited to one line per camera because if more line would exist, than more intersection would be generated with the epipolar line and 3D coordinates wouldn't be possible to calculate. The case when the line is parallel to the epipolar line is solved.
b. one arc in 2 or more cameras (mc_arcpt)
c. two arcs representing the left and right side of a distorted circle, find 3 points on the circle in all images using epipolar geometry

SLI.bas

```
' This is a "slicer" pseudolibrary used to re-segment a line
' based on new intervals. Please refer to "'Slicer' Algorithm"
' document (C=KJN 18 Mar 04) for more details.
'
' This library is especially good for making slices "on the
' fly" for a lot of little slices rather than building a huge
' list of (X,Y) locations in memory.
'
' Please refer to SLI_TEST.FRM for an example of how to
' use the routines in this library to do slicing (and the
' order in which to call routines).
'
' Created especially for Cognex bead inspection demo.
```

Simple interpolation linear functions.

Utl.bas

Matrix operations utilities (mmult(), transpose())

VB2.bas

```
' This module is a 2D calibration/computation library.
' Originally AUX_MOD.BAS, a new 2D library was finally created
' on December 4, 2002.
'
' Looking for special 2-camera TCP calibration algorithms?
' These (currently) are not normal 2D but a special
' application-specific thing whipped up for Twinsburg; please
' see the CCVRID-C project to find these algorithms (gve_tcp2()).
'
' Required AUDO structure for VB2 test CF files only:
'   ALL "Object containing ALL trained data" [3]
'     RBT "Robot 3D Coordinates" [1]
'       XYZ(1) "Layer 1" [5]
'         DAT(1) "Point 1" [1.0, 2.0, 3.0]
'          :     :      :   :
'         DAT(5) "Point 5" [13.0, 14.0, 15.0]
'     CAM(1) "Camera 1" [1]
'       UV2(1) "Layer 1" [5]
'         DAT(1) "Point 1" [1.0, 2.0]
'          :     :      :   :
'         DAT(5) "Point 5" [9.0, 10.0]
' Note that the number of cameras, layers, and points is
' specified explicitly at the appropriate objects.
'
' Format of the generated 2D calibration data (designed
' to be backward-compatible with pre-Dec.'02 versions
' of RELIABOT PC CCVRID):
'
'   CA2(1) "Calibration for camera #1"
'     DAT(1) "X-Y-Roll-Z" [1.0, 2.0, 6.0, 3.0]
'     DAT(2) "Rate Scale Yaw Pitch" [1.0, 1.0, 4.0, 5.0]
```

'

' Note: X-Y-Z-Yaw-Pitch-Roll refer to a 'to.cam'
' transformation. The parent object, kind, index, and
' description are created by the calling program. Also, the
' descriptions with the DAT objects are not created (but
' could be by the calling program if desired).
'

Cal2_flat() – simple 2D calibration in a flat plane only it uses 3 2D image points and 3 3D robot points (actually only x and y from robot points).
Cal2_c() 'advanced' (in Shafi words) 2D calibration from three points – limitations – surface perpendicular to the camera.
Cnv2uv() – calculate the back-projection of the point in the image (using the calibration calculated wiht cal2_c) – simple math operations: a point is transform with space to camera transformation and then is scaled.
Cnv2xyz() – calculate the projection of the vision point in the 3d space (same as cnv2uv)

BV3.bas

' This pseudolibrary implements 3D vision algorithms. It is
' a conversion of the 3D-related "gve" V+ algorithms from
' RELIABOT version 1.70. Most of the code dates back to 1994
' and 1996. This module created November 14, 2001.
'
' Suggested usage (for full computation):
' 1. Get 2D coordinates of the five dots (LOWER layer)
' for all 3 cameras
' 2. Sort the five dots (use vb3.x5sort()).
' 3. Get 3D coordinates of each dot.
' 4. Put 2D and 3D results into AUDO list structure.
' 5. Repeat steps 1-4 for UPPER layer.
' 6. Do calibration via repeated calls to vb5.cal3.f().
' Need three AUDO objects to store the final
' matrix for each camera.
'
' To compute for a single camera and single set of 6 points
' only, pick out the desired coordinates, then make a
' simple one-stop call to vb3.cal3_c(). You specify just

```
' one AUDO object to store the matrix computed; AUDO objects
' not necessary as inputs. Note: The magic point must be
' manually computed (if desired) via vb3.cal3_m().
'
' Required AUDO structure for vb3.cal3_f():
'   ALL "Object containing ALL trained data" [3]
'     RBT "Robot 3D Coordinates" [2]
'       XYZ(1) "Layer 1" [5]
'         DAT(1) "Point 1" [1.0, 2.0, 3.0]
'          :     :       :   :
'         DAT(5) "Point 5" [13.0, 14.0, 15.0]
'       XYZ(2) "Layer 2" [5]
'         DAT(1) "Point 1" [16.0, 17.0, 18.0]
'          :     :       :   :
'         DAT(5) "Point 5" [28.0, 29.0, 30.0]
'     CAM(1) "Camera 1" [2]
'       UV2(1) "Layer 1" [5]
'         DAT(1) "Point 1" [1.0, 2.0]
'          :     :       :   :
'         DAT(5) "Point 5" [9.0, 10.0]
'       UV2(2) "Layer 2" [5]
'         DAT(1) "Point 1" [11.0, 12.0]
'          :     :       :   :
'         DAT(5) "Point 5" [19.0, 20.0]
'     CAM(2) "Camera 2" [2]
'       <etc.>
'     CAM(3) "Camera 3" [2]
'       <etc.>
' Note that the number of cameras, layers, and points is
' specified explicitly at the appropriate objects. The
' ALL object (doesn't have to be of kind 'ALL') is the
' one that is passed to vb3.cal3_f().
'
' Format of the generated 3D calibration data:
'   CA3(1) "Matrix for camera #1" [1]
```

' DAT(1) "Row 1" [8.0, 7.0, 6.0, 5.0]
' DAT(2) "Row 2" [8.0, 7.0, 6.0, 5.0]
' DAT(3) "Row 3" [8.0, 7.0, 6.0, 1.0]
' DAT(4) "Magic Pt" [1.0, 2.0]
' DAT(5) "Sens for Alg 1-4" [1.2 1.2 1.2 1.2]
' DAT(6) "Sens for Alg 5-8" [1.2 1.2 1.2 1.2]
' DAT(7) "Sens for Alg 9-12" [1.2 1.2 1.2 1.2]
' DAT(8) "Sens for Alg 13-16" [1.2 1.2 1.2 1.2]
' DAT(9) "Sens for Alg 17-20" [1.2 1.2 1.2 1.2]
' The parent object, kind, index, and description are
' created by the calling program. Also, the descriptions
' with the DAT objects are not created (but could be
' by the calling program if desired).
'
' On Sensitivty: This is not a V+ conversion, but something
' NEW added February 2002. The best algorithm is shown as
' [1] above in the CA3 object. All sensitivities are stored
' for future reference in DAT(5) to DAT(9). Note that the
' sensitivity is stored for CAMERA 1 ONLY (cameras 2 and 3
' do not have this information because sensitivity is
' by its nature multi-camera).

X5sort() – sort a list of 5 points of the calibration target (if # of points <> 5 , the routine is not working).
Gauss_elim() – solves A*X=B
Cal3_c() – uses as input para,etrs a list of corresponding 3D and 2D points. The 3D coordinates are taught by a robot tooltip. It is a simple perspective calculation of a camera matrix using 6 calibration dots positions. The perspective matrix includes the camera internal parameters and camera extrinsic parameters but no distortion. The relation between the camera and robot coordinate system is not calculated. The points can be transformed to the robot base, but the camera has to be in the exact same position as at training time. A linear system of equations is used to find the perspectrive matrix (3x4).
Cal3_f() – same as Cal3_c but uses permutations to find the 'best' calibration.
Cal3_t() – tests how well a 2D point is predicted from a 3D points that was not used during the calibration.
Sen3_t() – tests sensitivity of 3D calibration for a pair of 3 cameras by choosing which component from each camera to use (x, y from each camera but not more than 3.).
Sen3_f() –
This routine is used to find the algorithmwith the least sensitivity and to reportjust what that sensitivity is.

The robot points used for 3D calibration (usually 10 points) are used as test points.
The points are converted to 2D for all three cameras, quirked, then computed back to 3D,
and a test is done to see how far they are off (this is the sensitivity). All 20 algorithms are tried to find the best one.
Xyz2uv – calculate the vision coordinates of a 3D point using the calibration calculated with cal3_c
C1c2_2xyz –
This program returns a 3D robot location for the2-dimensional points (u,v) of a "matched" pair. The U and V coordinates of the point from the first camera and the U *or* V coordinate from the second camera are used to perform the conversion. The involved cameras must have already been calibrated with a perspective matrix.
C1c2c3_2xyz – same as c1c2_2xyz
Pair2xyz() – same a c1c2_2xyz but it gives more permutations than c1c2_2xyz
Trip2xyz() – same a c1c2c3_2xyz but it gives more permutations than c1c2c3_2xyz
Uv2xyz() – image to space point transformation, z must be provided

VPS.bas

```
' This module contains subroutines to simulate V+ instructions,
' functions, and capabilities. For example, the ATAN2()
' function duplicates the functionality of ATAN2() in V+, and
' the COLON2() subroutine simulates the composition of
' 2D transformations.
'
' For this particular module only, we depart from the standard
' "all lowercase" naming convention of routines and use
' all uppercase. This is to 1) emphasize that these routines
' are part of the V+ simulation package, and 2) make the
' functions LOOK like they do in V+... they are all uppercase
' in V+.
'
' This module was originally a subset of AUX_MOD.BAS in
' CCVRID-C, but it has now been converted into a generic
' pseudolibrary which is developed independently and can be used
' in any project.
```

3D transformation utilities: euler to matrix and matrix to euler; distance between 2 3d points; add 2 3d transformations; orther 2d and 3d geometric utilities – some of them hardcoded.

Appendix B - Software Architecture and Engineering

Reliabot

The basic architecture of Reliabot is based on the concept of the CF file. CF files define the run-time configuration of the Reliabot UI, vision guidance and device interaction (robots, cameras, etc). When Reliabot is in Manual mode, the solution provider employs a set of wizards to generate the CF file. In many instances, the CF file may require further hand-editing. We'll return to this point in a moment. When Reliabot is in Auto mode, it loads a CF file, interprets it, establishes connections with devices, sets up the user interface and begins running the vision guidance solution.

It is difficult to discuss the software architecture of Reliabot without first describing the development model for the system. From the best evidence we have, new versions of Reliabot were produced for each solution that was developed. While several core components remained fixed (such as CF file parsing), each version was heavily customized for the solution in question. Customizations may have involved adding or altering software code, or hand-tweaking the contents of CF files. Most importantly, new versions were often created by copying the entire directory of source files and/or an existing CF file. As such, it is difficult to discern which code file represents the definitive version of a given Reliabot module. This presents some minor difficulties for determining the best code base from which to continue Reliabot development.

We have determined that the CF file syntax consists of 161 distinct commands, give or take a handful of special directives. Each command is indicated by a three letter acronym (e.g. DAT or CA3) and will have one or more numerical or text arguments, subcommands, etc. Documentation is available for a handful of commands. Learning any given undocumented command will require locating the software code that interprets the command arguments and reverse-engineering the code to determine the command syntax. Alternately, some commands may be reverse engineered by examining the wizards that emit CF file directives at configuration time. Regardless, fully documenting a single CF command would likely require 3-6 developer-hours for simple commands and 1-3 developer-days for complex commands. Documentation of the CF syntax is essential for internal development and support. Whether this documentation would be shared with end users and/or integrators is a business decision- withholding documentation helps to obfuscate Reliabot operation and limits the end user's support options. Providing documentation may reveal critical Reliabot IP but would enable users and integrators to provide their own internal support, reducing the demands on Braintech infrastructure.

Where CF files represent the configuration state of a particular solution, the software code represents the Reliabot program itself. Reliabot consists of approximately 97 code files defining 1056 subroutines. Like CF commands, most of the Reliabot modules are represented by three-letter acronyms (e.g. ado, ape, se2). There does not appear to be a 1-1 correspondence between modules and CF commands. Most of the source files contain some short documentation describing the module and revision history. For a complex, large-scale software system, Reliabot is relatively small. A great deal of the vision science is out-sourced to third party libraries such as Cognex VisionPro, and in addition, considerable solution logic is contained in the CF files. For many modules, documenting the source for internal use should be moderately straightforward. Others, however, contain upwards of 40,000 lines of code and will be extremely difficult to document.

One critical element for building high-performance, compute intensive applications, such as bin picking, is a robust multi-threading model. This enables one or more vision tasks to take place in parallel with user interface and robot motion tasks. While there is language support for multithreading in the Reliabot environment, the code was not developed with multithreading in mind, and due to the pervasive use of global variables it would be virtually impossible to refactor Reliabot to take advantage of multithreading.

Pros:
The concept of Reliabot as a solution engine.
Small, and relatively easy to document the basic source modules.
Configurable GUI based on CF file.
Obfuscated CF syntax may deter (but not prevent) reverse engineering by end user.
Emphasis on UI simplicity benefits the end user.

Cons:
CF commands are nearly entirely undocumented and require reverse-engineering.
Difficult to document the large, complex source modules.
Penchant for undocumented three letter acronyms is a developer's nightmare.
Lack of formal versioning system will present one-time obstacles for establishing a stable code base.
Ability to extend, enhance functionality is limited due to limited documentation (see also section 7).
Inadequate multithreading support.

eVF

Like Reliabot, eVF is a monolithic software application that provides vision guidance. eVF shares similar configuration and run-time functionality, and in addition encapsulates a significant investment in Braintech IP (SC3D, RBP, etc). The eVF analogue of the CF file is the eVF Workspace. The Workspace is configured using the eVF user interface and, with minor exceptions, is encoded such that reverse engineering is very difficult. More significantly, the eVF Workspace stores configuration parameters but no significant run-time logic related to vision science.

The eVF development model involves strict version control, a regular develop-test-release cycle and ongoing vigilance as to the quality of the finished product. eVF has benefitted from a rigorous testing regimen that focuses on both the quality of the vision science and end user usability. As a result, while the user is prevented from examining the internals of a workspace, the user-friendliness of the interface, coupled with extensive end-user documentation, enables them to solve minor configuration issues without relying heavily on Braintech support.

eVF development based on the concepts of object-oriented programming and modularity. This enables Braintech to expand the capabilities of the system to support new hardware devices and new vision guidance methodologies. However, the eVF code base is very large, consisting of approximately 2500 source files containing 580,000 lines of code. Rough estimates set the re-implementation cost in the neighbourhood of $20M-$30M. The size and complexity of the code base makes it impossible for any one developer or scientist to be familiar with everything. This can present difficulties when new features are added, and as a result new feature development is typically a team effort.

Pros:
User friendliness
Strong IP protection
Well-documented
Excellent revision control.
Rigorously Tested
Modular, object oriented.

Cons:
Monolithic
Very large, very complex.
Limited ability to customize the user interface.

Conclusions
The outstanding differences between the eVF and Reliabot architectures are that eVF is well-documented with a strong object-oriented methodology, and has benefitted from a focus on developing, testing, and maintaining a single software product on which all end-user solutions are based. There is one and only one eVF code base and new versions of eVF aim for backwards compatibility with previous commercial releases. By contrast, there will be considerable obstacles in identifying a stable code base, and building, testing and extending a core software product from the current Reliabot code. We believe that these are critical capabilities if Reliabot is to thrive as a provider of vision guidance solutions.

Appendix C - Robot Interfaces and Robot Programs

Reliabot
Reliabot in general communicates with robots by sending raw data over a serial connection, and having the robot programs interpret this raw data. This means that individual robots do not have specific communication interfaces, it is the robot programs that need to be created for each robot controller. The exceptions are Fanuc, which has a specific Ethernet interface, and Motoman, which uses the Motoman communication serial protocol to write data to numeric variables.

Reliabot robot program examples are available for seven robot types; ABB, Adept, Fanuc, Kawasaki, Motoman, Nachi, Staubli. ABB and Motoman programs were examined. The vision related code is in the form of routines that are added to perform three basic operations; Sending trigger from robot to Reliabot, receiving data from Reliabot, and offsetting points to be used for motion.

The adjustments to robot motion based on vision are done on a point by point basis. For instance, let's say there are two points, P1 and P2, that need to be moved to for picking a part. After vision is run, P1 and P2 are each offset by the X, Y, and Z values that come from Reliabot.

```
partPos =D_near_push;
partPos.trans.x:=serialReturn{1};
partPos trans x:=partPos trans.x+C_actual.trans.x;
partPos.trans.x:=partPos.trans.x-C_piktak_base1.trans.x;
partPos.trans.y:=serialReturn{2};
partPos.trans.y:=partPos.trans.y+C_actual.trans.y;
partPos.trans.y:=partPos.trans.y-C_piktak_base1.trans.y;
partPos.trans.z:=serialReturn{3};
partPos.trans.z:=partPos.trans.z+C_actual.trans.z;
partPos.trans z:=partPos.trans.z-C_piktak_base1.trans.z;      partPos.rot:=OrientZYX(serialReturn{6},serialReturn{5},serialReturn{4})
```

Communication is handled by sending and receiving strings through the serial port. When vision offsets are needed, the string "SHIFT M" + the model number is sent over the serial port to Reliabot.

```
Open "COM1:",serial_Comm\Write;
Write serial_Comm,"SHIFT M="+NumToStr(model_num,0);
Close serial_Comm;
Open "COM1:",serial_Comm\Bin;
ClearIOBuff serial_Comm;
WriteBin serial_Comm,carReturn,1;
Close serial_Comm;
```

Reliabot will return a string containing 6 comma separated values representing Tx, Ty, Tz, Rx, Ry, Rz. There is a routine on the robot which parses these values and places them into an array (serialReturn) which is used for offsetting points.

```
Open "COM1:",serial_Comm\Read;
serialLine:=ReadStr(serial_Comm\Time:=10);
Close serial_Comm;
TPWrite serialLine;
stringPlace2:=StrFind(serialLine,1,STR_WHITE);
FOR i FROM 1 TO stringLength DO
 stringPlace:=stringPlace2-1;
 stringPlace2:=StrFind(serialLine,stringPlace2+1,".1234567890-"\NotInSet)+1;
 len:=((stringPlace2-1)-(stringPlace+1));
 value:=StrPart(serialLine,stringPlace+1,len);
 TPWrite value;
 ok:=StrToVal(value,serialReturn{i});
ENDFOR
```

Pros:
Simple to implement on Reliabot side.
One interface supports multiple robots.
More robots currently available.

Cons:
So simple that functionality is limited. Advanced routines such as AutoCal, AutoTrain, Accutest are not possible.
Different solutions require customization of the data being sent.
Point by point offsets make it impossible to perform complex robot motions in 3D.
Setup and configuration of serial hardware is complicated and error prone.
No standard robot programs.
Slow communication speed.

eVF

Communication in eVF has been accomplished by individual communication components for each robot controller brand. This makes it possible to read and write directly to named variables on the robot side. Robot programs are created as Application Programming Interfaces that simplify the use of vision by robot programmers, and make the vision related code portable to any solution. Communication in the eVF case is not visible to the robot program and does not need to be run explicitly. The code below is an example of what is necessary to execute a runtime cycle.

```
!Call eVF to return the position of a part
bSuccess:=ViCalculatePose(nTrigger,4,FALSE);
```

There are four robots supported by eVF; ABB, Motoman, Kawasaki, Denso. There are a number of advanced functionalities that are supported for ABB(and to a lesser extent, Motoman). These include AutoCal, AutoTrain, Accutest, ReOrientation for increased accuracy, automated movement of robot to training position. Steps have been taken to create a generic robot interface for eVF that will minimize the amount of development needed to integrate a new robot. Full featured Kuka interface is nearing completion.

Pros
Hardware setup is simplified (and documented).
Rich interface allows for advanced setup and runtime procedures.
Robot programs are simplified.
Standardized robot programs reduce support burden.
Fast communication speed.

Cons
New robots require more up front development on eVF side.
Fewer robots currently available.

Conclusions:
The Reliabot communication method has the benefit of being able to support multiple robots from the same interface, and more robot brands have been integrated. Adding robots still requires developing robot programs to provide the other side of the communication link. The Reliabot communication interface is antiquated, slow, and very simplistic. The eVF communication interfaces are designed to provide advanced functionality and automated procedures that are not possible with Reliabot. While eVF does not currently offer as many robot interfaces as Reliabot, the path to arriving at a set of full featured and flexible robot communication interfaces will be much shorter by adding more robots to eVF, than by extending or recreating every interface in Reliabot.

Appendix D - Software Development Environment and Tools

Reliabot
Reliabot is coded in Visual Basic 6 (VB6, aka Classic Visual Basic). VB6, the final version of Visual Basic was released in 1998.

Mainstream support for VB6 ended on March 31, 2005, and Extended support ended on April 8, 2008.

Pros and Cons of VB6:

Pros:
Allows for rapid prototyping. (BUT so do modern .NET languages such as C#, BV.NET, Managed C++)
GUI development is fast and easy.
Language is easy to learn.
Can export ActiveX objects via COM (in theory, can integrate with eVF).

Cons:
Obsolete language, no longer MSFT-supported.
MSFT no longer sells the VB6 compiler/IDE- how to expand development without seats?
Performance lags that of C/C++. Probably also lags .NET.
Weak object oriented support.
Not suitable for real-time (e.g. real-time part tracking), due to garbage collection.
Not portable to other operating systems.
Lack of language support implies difficulties integrating new hardware (cameras, robots, etc).

eVF
By comparison, eVF is developed using a combination of C/C++ and C# (mostly C++). Newer projects are increasingly written in C#. eVF has a strong COM (component object model) dependency which facilitates .NET interop but limits portability.

Pros and Cons of the eVF development model (C/C++/C#/COM)

Pros:
Fully object oriented (ease of extensibility, modern development concepts, etc).
Internationally recognized language standards.
Language support can be expected to continue for another decade or more.
Access to WIN32 API and C programming == major performance advantage.
Ease of integration with new hardware.

Cons:
GUI development using WIN32 API is difficult.
COM dependency ties us to WIN32 platform.
COM may not have long-term support.
Easy to write 'spaghetti code': extremely complex software systems.

Further reference:
http://en.wikipedia.org/wiki/Visual_basic
http://en.wikipedia.org/wiki/C++
http://en.wikipedia.org/wiki/C_(programming_language)
http://en.wikipedia.org/wiki/Component_Object_Model
http://en.wikipedia.org/wiki/Win32

Conclusions:

Continuing Reliabot support using VB6 presents a serious challenge- lack of vendor support, lack of available development licenses, and a lack of modern programming language advances all represent red flags against developing and maintaining a large-scale software. There is no easy out in this respect, since re-writing Reliabot in a modern language would represent thousands of developer-hours. While eVF is also built on aging technology, the core C/C++ languages will be supported for decades to come, and our ability to inter-operate with C# can smooth the path to modernization. The main concern with eVF is its dependency on COM and marriage to the Windows platform- a concern that should be resolved with the roll-out of eVF 7.0 (BVGL/BVSL).